Final Report
February, 1996
NASA - Ames Grant NCC2-550

Research on Computer Systems Benchmarking

Principal Investigator:
Professor Alan Jay Smith
Computer Science Division
EECS Department, Electronic Research Laboratory
University of California
Berkeley, CA 94720-1776
smith@cs.berkeley.edu 510-642-5290

This grant addresses the topic of Research on Computer Systems Benchmarking and is more generally concerned with performance issues in computer systems. This report reviews work in those areas during the period of NASA support under this grant.

The bulk of the work performed under this grant was done by a graduate student, Rafael Saavedra-Barrera, who received his Ph.D. during the period of funding. (Ph.D., February, 1992, "CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking", (Computer Science Division Technical Report UCB/CSD 92/684), MS, September, 1988, "Machine Characterization and Benchmark Performance Prediction" (UC Berkeley Computer Science Division Technical Report 88/437). Saavedra-Barrera was the winner of the David Sakrison award for the best Ph.D. thesis in the EECS Department in 1991-92.)) His research concerned benchmarking and analysis of CPUs, compilers, caches and benchmark programs. The first part of this work concerned the issue of benchmark performance prediction. ("Machine Characterization Based on an Abstract High Level Language Machine", Rafael Saavedra-Barrera, Alan Jay Smith and Eugene Miya, IEEE Transactions on Computers, special issue on Performance Evaluation, December, 1989, 38, 12, pp. 1659-1679.) Runs of a benchmark or a suite of benchmarks are inadequate to either characterize a given machine or to predict the running time of some benchmark not included in the suite. Further, the observed results are quite sensitive to the nature of the benchmarks, and the relative performance of two machines can vary greatly depending on the benchmarks used. In this first paper, we reportecd on a new approach to benchmarking and machine characterization. The idea is to create and use a machine characterizer, which measures the performance of a given system in terms of a Fortran abstract machine. Fortran is used because of its relative simplicity and its wide use for scientific computation. The analyzer yields a set of parameters which characterize the system and spotlight its strong and weak points; each parameter provides the execution time for some primitive operation in Fortran. We presented measurements for a large number of machines ranging from small workstations to supercomputers. We then combined these measurements into groups of parameters which relate to specific aspects of the machine implementation, and used these groups to provide overall machine characterizations. We also defined the concept of pershapes, which represent the level of performance of a machine for different types of computation. We introduced a metric based on pershapes that provides a quantitative way of measuring how similar two machines are in terms of their performance distributions. This metric was related to the extent to which pairs of machines have varying relative performance levels depending on which benchmark is used.

In another paper to come out of his research, ("Performance Characterization of Optimizing Compilers", Rafael Saavedra-Barrera and Alan Jay Smith, IEEETSE, July, 1995, vol. 21, no.

7, pp. 615-628), Saavedra analyzed compiler performance. Optimizing compilers have become an essential component in achieving high levels of performance. Various simple and sophisticated optimizations are implemented at different stages of compilation to yield significant improvements, but little work has been done in characterizing the effectiveness of optimizers, or in understanding where most of this improvement comes from. In this paper we studied the performance impact of optimization in the context of our methodology for CPU performance characterization based on the abstract machine model. The model considered all machines to be different implementations of the same high level language abstract machine; in previous research, the model has been used as a basis to analyze machine and benchmark performance. In this paper, we: 1) showed that our model can be extended to characterize the performance improvement provided by optimizers and to predict the run time of optimized programs; 2) measured the effectiveness of several compilers in implementing different optimization techniques; and 3) analyzed the optimization opportunities present in the Fortran SPEC and other benchmarks.

Benchmark programs are analyzed in another paper by Saavedra ("Analysis of Benchmark Characteristics and Benchmark Performance Prediction", Rafael Saavedra-Barrera and Alan Jay Smith, Technical Report UCB/CSD-92-715, December, 1992, submitted for publication). Standard benchmarking provides the run times for given programs on given machines, but fails to provide insight as to why those results were obtained (either in terms of machine or program characteristics), and fails to provide run times for that program on some other machine, or some other programs on that machine. We have developed a machine-independent model of program execution to characterize both machine performance and program execution. By merging these machine and program characterizations, we can estimate execution time for arbitrary machine/program combinations. Our technique allows us to identify those operations, either on the machine or in the programs, which dominate the benchmark results. This information helps designers in improving the performance of future machines, and users in tuning their applications to better utilize the performance of existing machines. Here we applied our methodology to characterize benchmarks and predict their execution times. We presented extensive run-time statistics for a large set of benchmarks including the SPEC and Perfect Club suites. We showed how these statistics can be used to identify important shortcomings in the programs. In addition, we gave execution time estimates for a large sample of programs and machines and compare these against benchmark results. Finally, we developed a metric for program similarity that makes it possible to classify benchmarks with respect to a large set of characteristics.

Saavedra also considered the effect of the memory hierarchy in: ("Measuring Cache and TLB Performance and Their Effect on Benchmark Run Times", Rafael Saavedra-Barrera and Alan Jay Smith, IEEE TC, October, 1995, 44, 10, pp. 1223-1235.) In previous research, we developed and presented a model for measuring machines and analyzing programs, and for accurately predicting the running time of any analyzed program on any measured machine. That work is extended in this paper by: (a) developing a high level program to measure the design and performance of the cache and TLB for any machine; (b) using those measurements, along with published miss ratio data, to improve the accuracy of our run time predictions; (c) using our analysis tools and measurements to study and compare the design of several machines, with particular reference to their cache and TLB performance. As part of this work, we described the design and performance of the cache and TLB for ten machines. The work presented in this paper extends a powerful technique for the evaluation and analysis of both computer systems and their workloads; this methodology is valuable both to computer users and computer system designers.

A summary of some of the early work on this project appears in "Performance Prediction by Benchmark and Machine Analysis", Rafael Saavedra-Barrera and Alan Jay Smith, Computer Science Division Technical Report UCB/CSD 90/607, December, 1990. That paper has been revised and updated, and should appear shortly as a book chapter. In this paper, we present a new

methodology for CPU performance evaluation based on the concept of an abstract machine model and contrast it with benchmarking. The model consists of a set of abstract parameters representing the basic operations and constructs supported by a particular programming language. The model is machine-independent, and is thus a convenient medium for comparing machines with different instruction sets. A special program, called the machine characterizer, is used to measure the execution times of all abstract parameters. Frequency counts of parameter executions are obtained by instrumenting and running programs of interest. By combining the machine and program characterizations we can and do obtain accurate execution time predictions. This abstract model also permits us to formalize concepts like machine and program similarity. A wide variety of computers, from low-end workstations to high-end supercomputers, have been analyzed, as have a large number of standard benchmark programs, including the SPEC scientific benchmarks. We present many of these results, and use them to discuss variations in machine performance and weaknesses in individual benchmarks. We also explain how the basic model can be extended to account for the effects of compiler optimization, memory hierarchy, and vectorization. We also indicate, when appropriately, how these factors affect our ability to predict the execution time of programs. More details are given in the appropriate references.

The work by Saavedra was continued into the domain of parallel and vector machines by Stephen Von Worley. (Steven Von Worley, MS, May, 1995, "Microbenchmarking and Performance Prediction for Parallel Computers"; "Microbenchmarking and Performance Prediction for Parallel Computers", (Stephen Von Worley and Alan Jay Smith), Technical Report UCB/CSD-95-873, May, 1995, submitted for publication.) In that work, we extended the earlier work to parallel computers. We described a portable benchmarking suite and performance prediction methodology which accurately predicts the run times of Fortran 90 programs running upon supercomputers. The benchmarking suite measures the optimization capabilities of a given Fortran 90 compiler, execution rates of abstract Fortran 90 operations, and the processing characteristics of the underlying architecture as exposed by compiler-generated code. To predict the run time of an arbitrary program, we combine our benchmark results with dynamic execution measurements, and augment the resulting prediction with simple factors which account for overhead due to architecture-specific effects, such as remote reference latencies. We measured two supercomputers: a dedicated 128-node TMC CM-5, a distributed memory multiprocessor, and a 4-node partition of a Cray YMP-C90, a tightly-integrated shared memory multiprocessor. Our measurements show that the performance of the YMP-C90 far outstrips that of the CM-5, due to the quality of the compilers available and the architectural characteristics of each machine. To validate our prediction methodology, we predicted the run time of five interesting kernels on these machines; nearly all of the predicted run times are within 50-percent of actual run times, much closer than might be expected.

In addition to the work described above, related work has also benefitted to some extent by NASA support. A student named Jeff Gee finished his Ph.D. in 1993 ("Analysis of Cache Performance in Vector Processors and Multiprocessors"). The first part of that work addressed the issue of the use of caches in vector processors. In the paper "The Performance Impact of Vector Caches", (Gee and Smith, Proc. 25'th Hawaii Intl. Conf. on System Sciences, January, 1992, Hawaii, Volume I, pp. 437-448), we considered whether vecctor supercomputers should have caches. Cache memories have not been used for vector supercomputers, as far as we know, because of a belief that program behavior in relevant workloads was such as to preclude efficient cache operation. It has been possible to make efficient use of such machines by carefully programming around the resulting long memory delays, although unmodified, "dusty-deck" code usually performs poorly. In related research, we have found that hit ratios are high for large caches in processors with vector workloads. In this paper, we addressed the specific issue of the direct effect of cache memory on vector processor performance. The issue in processor design is machine performance, of which the hit ratio of the cache is only one determinant. In this paper,

we simulated three vector processors, the designs for which are derived from expected technology changes applied to the Ardent Titan. Our simulator was an accurate timing model incorporating the necessary aspects of the design of the cache and memory system. We found that current trends in memory and processor performance will lead to increasingly severe memory speed and bandwidth limitations. Either of two designs using large cache memories (2MB, 4MB) on the average double processor performance relative to the design without a cache. Hit ratios for almost all of the programs used for trace driven simulation, drawn from real Ardent workloads, are over 99%. Based on the work presented here and elsewhere, we recommend that future supercomputers incorporate cache memories.

The second vector cache study with Gee ("The Effectiveness of Caches for Vector Processors", Jeffrey Gee and Alan Jay Smith, Proc. Int. Conf. on Supercomputing, Manchester, England, July 11-15, 1994, pp. 333-343.) more directly studied the behavior of vector workloads. Vector processors have typically used vector registers, interleaved memory, and pipelined access to data to provide sufficient memory system performance. Caches have been used mainly for instructions and scalar data, while vectors are usually uncached, presumably partially because of the belief that there is insufficient vector locality in these workloads. In this study we used memory address traces from an Ardent Titan to examine both reference locality and cache performance in a vector processing environment. Many of the Titan traces are from real vectorized applications which reference large amounts of data. We found that vector references contain somewhat less temporal locality, but large amounts of spatial locality compared to instruction and scalar references. Cache miss ratios were found to be comparable to those measured and published previously for various non-vectorized workloads. We provided analyses of trace behavior with regard to parameters of interest to cache designers. Calculations based on our measured miss ratios indicated that caches will improve average access times, which in turn can be expected to translate into significant improvements in machine performance. Arguments suggesting otherwise were discussed and considered.

The second part of Gee's research concerned cache memory design for multiprocessors. In the first paper with Gee, ("Analysis of Multiprocessor Memory Reference Behavior", Jeffrey Gee, Alan Jay Smith, Proc. ICCD'94 (IEEE Intl. Conf. on Computer Design: VLSI in Computers and Processors), Cambridge, MA, October 10-12, 1994, pp. 53-59), we analyzed multiprocessor memory reference behavior. Shared-memory multiprocessors can provide impressive performance at reasonable costs, although private caches are usually needed to alleviate the potential bottleneck at shared memory. These private caches in turn require the use of *cache-consistency (coherency) protocols*, whose performance is a strong function of the reference behavior within multiprocessor applications. In this paper we characterized the memory reference behavior in a wide variety of scalar and vector multiprocessor address traces from production workloads. This analysis was for the purpose of estimating and improving the performance of cache-consistency protocols. Our analysis extended previous results in the literature by performing a wider variety of analyses, and analyzing a larger and more diverse set of multiprocessor traces, including a production vector workload. We found wide differences between the sharing behavior observed in vector and scalar applications. Compared to scalar programs, vector programs reference shared data more frequently and contain larger amounts of *processor locality*, the tendency for shared data to be used by only one processor over periods of time. Write sharing by different processors over short intervals are infrequent in one workload but frequent in another. This implies that sequentially-consistent programming models will remain necessary unless applications are recoded to avoid such reference patterns.

The second MP-cache paper with Gee, ("Evaluation of Cache Consistency Algorithm Performance", Jeffrey Gee and Alan Jay Smith, Proc. Mascots'96 (Intl. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems) Conference, pp.

236-249, February 1-3, 1996, San Jose, CA.), presented the results of extensive simulations of multiprocessor cache consistency algorithms. As more computer systems turn to multiprocessing for improved performance, additional research is needed to evaluate and improve the performance of cache consistency protocols. In this study, we used trace-driven simulation to examine the performance of several consistency protocols, including some new adaptive protocols which have not been examined in prior research. This study used a wider variety of traces than have been previously analyzed, including some production applications from a vector mini-supercomputer system, and presented a wider variety of analyses than have been previously shown for a given workload. We found that the sharing characteristics of application programs have a large bearing on the relative performance of the different protocols. *Update-based* protocols outperform *invalidate-based* protocols when accesses to shared data are highly interleaved among different processors (fine-grain sharing), while invalidate-based protocols are superior if one processor performs all accesses to shared data over long periods of time (coarse-grain sharing). Adaptive protocols provide the best overall performance across all applications; we present a new protocol called Update-Once, which yields the highest average performance. In even the best cases, however, estimated processor utilizations are unacceptably low due to the overhead to maintain consistent caches. To extract good performance from multiprocessor systems, existing application programs must be recoded to reduce sharing between processors.

Gee also was a collaborator in a paper in which we measured the miss ratios for the SPEC benchmarks: ("Cache Performance of the SPEC Benchmark Suite" (Jeffrey Gee, Mark Hill, Dionisios Penvmatikatos, Alan Smith), IEEE MICRO, 13, 4, August, 1993, pp. 17-27. The SPEC benchmark suite consists a set of public-domain, non-trivial programs that are widely used to measure the performance of computer systems, particularly those in the Unix workstation market. These benchmarks were expressly chosen to represent real-world applications and were intended to be large enough to stress the computational and memory system resources of current-generation machines. The extent to which the *SPECmark* (the figure of merit obtained from running the SPEC benchmarks under certain specified conditions) accurately represents performance with live real workloads is not well established; in particular, there has been some question whether the memory referencing behavior (cache performance) is appropriate. In this paper, we presented measurements of miss ratios for the entire set of SPEC benchmarks for a variety of CPU cache configurations; this study extends earlier work that measured only the performance of the integer (C) SPEC benchmarks. We found that instruction cache miss ratios were generally very low, and that data cache miss ratios for the integer benchmarks were also quite low. Data cache miss ratios for the floating point benchmarks were more in line with published measurements for real (i.e. non-benchmark, non-synthetic) workloads. We believe that the discrepancy between the SPEC benchmark miss ratios and those observed elsewhere is partially due to the fact that the SPEC benchmarks are all almost exclusively user state CPU benchmarks run until completion as the single active user process. We therefore believe that SPECmark performance levels may not reflect system performance when there is multiprogramming, time sharing and/or significant operating systems activity.

Another student, John Tse, looked at the issue of prefetching into CPU cache memories. (John Tse, MS, June, 1995, "Performance Evaluation of Cache Prefetching Strategies"; "Performance Evaluation of Cache Prefetch Implementation", John Tse and Alan Jay Smith, Technical Report UCB/CSD-95-873, June, 1995, submitted for publication.) Prefetching into CPU caches has long been known to be effective in reducing the cache miss ratio, but implementations of prefetching have been unsuccessful in improving CPU performance. The reasons for this are that prefetches interfere with normal cache operation by making cache address and data ports busy, the memory bus busy and the memory banks busy, and by not necessarily being complete by the time that the prefetched data is actually referenced. In this paper, we presented the results of a very detailed cycle by cycle trace driven simulation of a uniprocessor memory system, in which

we vary several relevant architectural parameters in order to determine when and if prefetching is useful. We found that in order for prefetching to actually improve performance, the address array needs to be double ported, and the data array needs to either be double ported or fully buffered. It is also very helpful for the bus to be reasonably wide, bus transactions to be split and main memory to be interleaved. Under the best circumstances, i.e. with a significant investment in extra hardware, prefetching can significantly improve performance.

A student named Jeff Rothman has implemented a parallel program tracer, and is collecting traces from a variety of workloads. He is currently concentrating on the design of sector caches for uni- and multiprocessor machines. Our plans are to do the following: (a) Characterize the performance of sector caches, and determine optimal sector cache designs; such designs are particularly applicable for microprocessor based systems with on-chip tag storage and off-chip data storage. (b) Extend the study of sector caches to the multiprocessor case. (c) Evaluate the effectiveness of a new sector cache design that I have invented. (d) Study the effect of changes in the source code on sharing patterns and the effectiveness of consistency algorithms. Rothman spent fall of 1993 visiting at Siemens (Munich), where he developed a code analyzer which can be used for code reorganization to minimize bus traffic in MP systems.

In work by Chris Perleberg ("Branch Target Buffer Design and Optimization", Chris Perleberg and Alan Jay Smith, IEEETC, 42, 4, April, 1993, pp. 396-412), we studied Branch Target Buffers. A Branch Target Buffer (BTB) can reduce the performance penalty of branches in pipelined processors by predicting the path of the branch and caching information used by the branch. This paper discussed two major issues in the design of BTBs, with the goal of achieving maximum performance with a limited number of bits allocated to the BTB implementation. First is the issue of BTB management - when to enter and discard branches from the BTB. Higher performance can be obtained by entering branches into the BTB only when they experience a branch taken execution. A new method for discarding branches from the BTB was examined. This method discards the branch with the smallest expected value for improving performance, outperforming the LRU strategy by a small margin, at the cost of additional complexity. The second major issue discussed was the question of what information to store in the BTB. A BTB entry can consist of one or more of the following: branch tag (i.e. the branch instruction address), prediction information, the branch target address, and instructions at the branch target. A variety of BTB designs, with one or more of these fields, were evaluated and compared. This study was then extended to multilevel BTBs, in which different levels in the BTB have different amounts of information per entry. For the specific implementation assumptions used, multi-level BTBs improved performance over single level BTBs only slightly, also at the cost of additional complexity. Multi-level BTBs may, however, provide significant performance improvements for other machines implementations. Design target miss ratios for BTBs were developed, so that the performance of BTBs for real workloads may be estimated.

A new graduate student, Winston Hsu, has just started work on branch prediction. In previous work (with John K-F Lee and Chris Perleberg), we studied prediction algorithms for branches, and the design of branch target buffers. That work, and other work in the field, has not sufficiently considered two items that we plan to study. First, we would like to look at the effectiveness of branch prediction if one has access to the source code. In particular, one may be able to compute the branch direction earlier in the source code and/or compute an effective hint. Second, with superscalar or VLIW machines, extra hints or precomputations can often be done for free, using otherwise unoccupied slots in the pipelines. Given the much higher penalty for unpredicted branches in such machines, we believe that there is a large potential payoff to such a study.

Another student, Ricki Blau, also finished her Ph.D. during the period of this grant. (Ricki Blau, Ph.D., December, 1992, "Performance Evaluation for Computer Image Synthesis Systems"). Her dissertation applied performance analysis to the problem of computing complex-

three dimensional images. First, it identifies factors that affect the cost of image synthesis and characterizes the complexity of realistic images. Four categories of performance factors are defined: scene characteristics, viewing specifications, rendering parameters and the computing environment. This classification provides a framework for discussing image complexity and designing performance experiments. The complexity of several complex images from an actual animation workload is described in detail. A methodology is presented for the construction of reproducible and controllable performance measurement experiments. To measure the performance of a rendering system, an experimenter provides a set of test data, including image specifications. The dissertation describes a portable tool that generates test cases, varying the scene characteristics and viewing specifications under the control of a set of parameters. This model generator has been implemented for two different rendering systems. Its test cases have been used to detect performance differences between the two systems and to evaluate the effects of varying the scene characteristics. The last part of the dissertation addresses the workload partitioning problem for MIMD rendering systems. A simple, low-overhead adaptive algorithm balances the workload effectively on a 16-node rendering accelerator. The algorithm uses the rendering time observed for one frame to predict costs for the next frame. The resulting cost estimates can be used by a second algorithm to divide the work among the available processing nodes. The cost estimates are approximate, but are obtained with little overhead. The net result is an improvement of thirty to eighty percent over the previous load balancing schemes for production quality rendering. An analysis of several competing schemes demonstrates that tradeoffs between balancing the load and preserving locality are a key consideration in the design of a parallel rendering system.

Jay Lorch finished an MS on a study of power consumption in the Apple Powerbook. (Jacob Lorch, MS, December, 1995, "A Complete Picture of the Energy Consumption of a Portable Computer"). A paper on this work is in preparation: High battery lifetime is important to the usability and acceptance of portable computers. In order to develop strategies to minimize power consumption, designers need a good picture of the total power consumption of a system. For this purpose, we indicated the power consumptions of a set of portable computers and how they are broken down among the components of those computers. Then, we use user profiles to show how the use of power-saving features currently implemented serves to reduce these power consumptions by 41--66. We also show how these power-saving features affect the breakdown of overall power consumption, so that we can evaluate how successful certain new software techniques and hardware changes would be at reducing power consumption. The results of this paper point out the most promising avenues for further work in the reduction of power consumption, and indicate some strategies that can provide an immediate power benefit to the class of machines studied.

In work principally carried out by a graduate student, Vigyan Singhal, we collected and studied traces from database systems. We were mainly concerned with the issue of locking and concurrency control. Concurrency control is essential to the correct functioning of a database due to the need for correct, reproducible results. For this reason, and because concurrency control is a well formulated problem, there has developed an enormous body of literature studying the performance of concurrency control algorithms. Most of this literature uses either analytic modeling or random number driven simulation, and explicitly or implicitly makes certain assumptions about the behavior of transactions and the patterns by which they set and unset locks. Because of the difficulty of collecting suitable measurements, there have been only a few studies which use trace driven simulation, and still less study directed toward the characterization of concurrency control behavior of real workloads. In a paper written with Singhal ("Characterization of Contention in Real Relational Databases" Technical Report UCB/CSD-94-801, Computer Science Division, UC Berkeley, March, 1994, to appear, VLDB Journal), we present a study of three database workloads, all taken from IBM DB2 relational database systems running commercial applications in a

production environment. This study considers topics such as frequency of locking and unlocking, deadlock and blocking, duration of locks, types of locks, correlations between applications of lock types, two-phase vs. non-two-phase locking, when locks are held and released, etc. In each case, we evaluated the behavior of the workload relative to the assumptions commonly made in the research literature, and discuss the extent to which those assumptions may or may not lead to eroneous conclusions. We also presented a simple mathematical model which predicts the frequency of blocking to be expected in these workloads, and compare those predictions to the observed frequency. Singhal received his M.S. degree (under my supervision) in Spring, 1994. He finished his Ph.D. (in the area of CAD) in 1995.

In work principally carried out by a graduate student, Barbara Tockey Zivkov, an extensive study of disk caching performance is in progress. This study presents extensive analysis of disk traces taken from a variety of real, production computer systems, including a bank (Security Pacific), a transportation company (Crowley Maritime), a telecommunications company, an oil company (Gulf Oil), Monarch Marking company, and two other large corporations (who prefer to remain anonymous). The systems traced include both IBM and Honeywell mainframe systems, and include both normal operating systems traces and database systems traces. Some of the latter were obtained with the help of researchers at IBM Almaden research laboratory. The paper in preparation (which should be finished by spring, 1996) characterizes the traces, and then studies disk caching performance under a variety of algorithms. Zivkov is expected to finish her MS this spring.

Currently in progress is an effort (by graduate students Min Zhou and Jay Lorch) to collect traces from PC systems (both Intel X86 based PCs and Apple computers). These traces are being collected for two reasons. First, we are in the early stages of a research project in the area of algorithms for power management in portable computers. The traces collected will contain information on user input, disk activity, and application program activity. With regard to I/O systems, we expect the traces to be used for two purposes. First, we will be using them to study algorithms for minimizing disk power consumption. We can do this by determining when the disk is likely to be idle and thus turning it off. We can also improve performance by successfully improving disk caching performance (fewer disk I/Os) and by prefetching and retaining in semiconductor storage those portions of the disk address space that are likely to be needed in the near future. In addition, we expect to use the traces to study the design of disk caches in the PC environment. This is a subject of great industrial interest, but it doesn't seem to have been addressed by the disk or PC industries.